JOURNAL OF SOFTWARE MAINTENANCE: RESEARCH AND PRACTICE
J. Softw. Maint: Res. Pract. 2000; **12**:1–23

Practice

Preventive actions for residual Y2K defects: a case study

Michelangelo Interesse*,†

Netsiel S.p.A., TelecomItalia-Finsiel Group, 70124 Bari, Italy

SUMMARY

This paper deals with some of the precautions that have to be, or should have been taken for MIS systems in order to lower the risk level related to residual Y2K problems. In particular, it focuses on those proactive actions effective after December 1999 to prevent possible failures of information systems and consequent damage from residual Y2K defects. After a short description of the Y2K problem domain, it highlights the importance of defining and rehearsing a specific contingency plan, incorporating the possible actions to fulfill the due diligence requirements. Among these, a key role is played by preventive actions like Independent Verification and Validation (IV&V), System Future Test (SFT) and Time Horizon of Events (THE). The case study covers some of the relevant aspects of an IV&V project addressing a major application. Processes and tools are briefly described and the final outcomes with supporting data are discussed. At the end of the case study the benefits gained are stated. Evidence is provided of the synergic effects resulting from the combination of several preventive actions as pre-defined steps in a contingency plan. The paper ends with some lessons learned. Copyright © 2000 John Wiley & Sons, Ltd.

KEY WORDS: year 2000; Y2K; IV&V; compliance; contingency; verification; validation

1. INTRODUCTION

Over the last two years, most of the people involved in the IT business have started considering the date 1 January 2000 as the day when computers and electronically controlled devices could stop working, with serious risks for human life. As a matter of fact, the millennium countdown has reached its end—at least from the computers' point of view—and we can say that neither the more catastrophic expectations nor the more ordinary ones have been fully met. Now that the date has rolled over, most people no longer seem to care about the millennium Y2K bug, a position that is at least unwise. This situation is mainly due to the erroneous belief that the Y2K bug was an undeferrable problem that would reveal itself fully at a single precise point in time: midnight on 31 December 1999.

*Correspondence to: Michelangelo Interesse, Netsiel S.p.A., via S. Dioguardi 1, 70124 Bari, Italy.
†Email: m.interesse@netsiel.it

Received 28 October 1999
Revised 19 November 1999

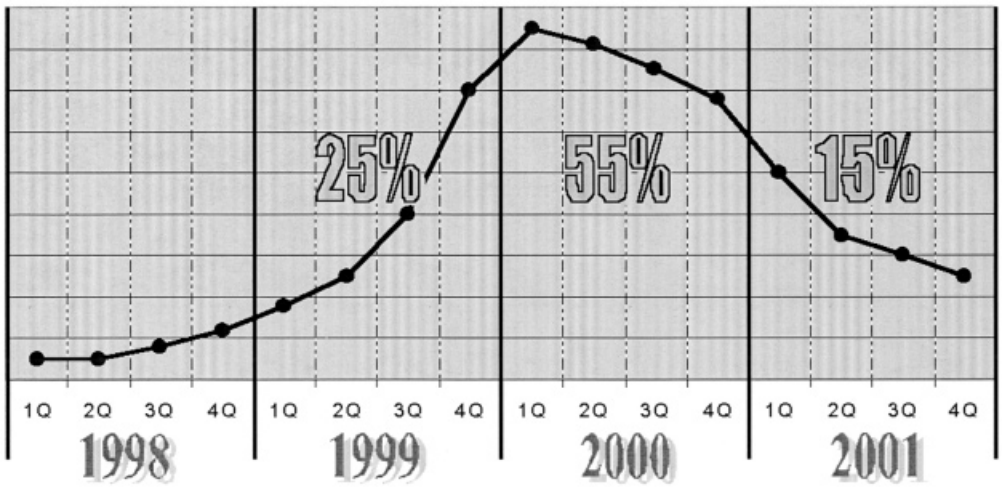


Figure 1. Distribution of potential Management Information System failures.

On the contrary, as shown by the results of specific research in this area [1], we need to consider the Y2K bug as a problem that goes far beyond such a reference date. In fact, at least for Management Information Systems (MIS) applications that govern worldwide business transactions, Y2K anomalies from remaining Y2K software defects or errors will manifest themselves in almost unpredictable ways for a period spanning more than a year (see Figure 1). During that span of time, residual Y2K anomalies will have a variable probability and damage potential, depending upon the specific business area or the industrial sector addressed by the underlying software systems.

In fact, 30–50% of global companies are going to experience some failures in their information and computing systems during the first quarter of 2000. In particular, at the century rollover a considerable percentage of failures was localized in the *embedded-systems* firmware or hardware—i.e., real-time clocks, microprocessors, microcontrollers, network and telecommunication devices, etc. Such failures represent the most dangerous ones for *human safety*, because of the physical side-effects they sometimes involve.

As for the transaction systems in MIS, most of the Y2K related failures are expected to be spread throughout the year 2000. These are considered to be very dangerous for *business safety*, especially when they affect the business-critical systems of an enterprise.

Some estimations [2] report that 5–9% of Y2K remediated and tested code still has remaining (residual) defects likely to cause failures. These failures may appear in a number of combinations and permutations that fall into the following categories:

- *abends*: transactions or batch processes fail completion, stopping the normal application execution;

- *incorrect computations*: transactions or batch processes produce incorrect results, that in turn are stored in the database and may propagate so as to affect other inherently compliant systems; and
- *external contamination*: databases are infected by erroneous data coming from external sources—e.g., non-compliant systems, unchecked input procedures, archived historical data, etc.

Normally, an extremely low percentage of MIS failures manifest themselves through system faults (abends); moreover, they are usually among the easier to discover, diagnose and fix. Conversely, most of the faults in the other two categories are discovered only days or weeks after they really happen. This makes detecting and fixing such software defects slow and expensive because the defects are usually very complex. In addition, recovery processes are sometimes needed to fix affected data.

Last year, over 50% of the companies started their testing process in the second quarter of 1999 or later [3]. Thus, they had insufficient time to apply all the required due diligence in performing such a critical task. However, the most conscientious and diligent Information Technology (IT) managers, when out of time but still with some budget remaining, decided to apply several compliance verification approaches in parallel—e.g., boundary test, future test, IV&V, etc. This allowed them to exploit more fully the time remaining and reduce the risk of failures. However, it seems reasonable to assume that the majority of the companies had not enough time or budget resources before 1 January 2000 to verify fully the Y2K compliance of all their systems and applications. So we should expect that even an extremely low percentage of latent defects still present in the executing code may cause significant problems. These problems from residual defects could affect the operations of other enterprises and their business partners or companies in their supply chains.

Such concerns are further sharpened by the consideration that Y2K problems have not usually been addressed off-line in an isolated sand-box, but instead been tossed in with the ongoing ordinary maintenance activities (corrections and enhancements). In this situation, we should not be surprised when we see an application starting from being non-compliant, being worked on to become compliant, and then slipping back into a non-compliant status in a relatively short period of time, possibly even including the century rollover.

Finally, some data and/or applications that have been frozen for several months and never adequately tested in the year 2000 environment may be used in execution later this year, potentially producing unsuitable results.

2. Y2K CONTINGENCY ACTIONS

Since we are now experiencing, day by day, all the facets of this scenario, we are led to realize the importance of having a contingency plan in order to soften the effects on a company's business triggered by incomplete Y2K compliance. As agreed by most of the analysts, a complete and well rehearsed contingency plan will be necessary during the year 2000, even for the best prepared company, so as to prevent, or at least to reduce, the impact of residual Y2K disruptions.

In particular, every Y2K contingency plan has to take into account all the following matters:

- business continuity and disaster recovery [4];
- risk management;
- utility disruptions;
- litigation and related legal actions;

- supply chain management; and
- front- and back-office operations.

Moreover, every Y2K contingency plan has to concentrate on *high-risk* and *mission-critical* systems and applications. In brief, these are the systems the company relies upon. To stop, disable or impair them will cause a sensible decrease in the production and distribution of the company's goods and services, or a lowering in the service-level quality, that in turn is likely to affect the customer portfolio and the expected revenue.

In order to focus on *mission-critical* applications, an attempt needs to be made to assess all the IT applications making up the company's software assets. This allows the company to detect, categorize and prioritize the mission-critical applications, thus providing a path for setting up a suitable contingency plan.

The actions that a contingency plan has to consider, in order to fulfill the due diligence requirements, can be classified into two categories, depending on the time when they need to be executed, prior to or after anomalies come into view:

- *preventive actions*, to anticipate and defuse the anomalies—e.g., System Future Test (SFT), Independent Verification and Validation (IV&V), Time Horizon of Events (THE) estimation, etc.; and/or
- *consequential actions*, which need to be preventively defined and then executed in response to system anomalies, whenever and wherever they appear, in order to remove them and to return the system to its normal operational mode, in as short a time as possible—e.g., automatic failure notification, error analysis and fixing, debugging and tracing, effective backup/restore procedures for critical data, alternative/backup processes, business continuity procedures, etc.

The latter is generally considered to play a key role in any contingency plan, but it relies, among other things, on the assumption that a very limited (as low as possible) number of failures comes into view in a reasonably short period of time. A further discussion about consequential actions is beyond the scope of this paper.

Unfortunately, the above-mentioned assumption does not prove to be always true. This is the main reason why analysts strongly recommend some preventive actions in order to reduce as much as possible the risk of failures. Among these, IV&V has gained a favorable reputation during recent months, mainly due to its advantageous ratio of achieved benefits to implementation costs.

The rest of this paper focuses on *preventive actions* as part of a general contingency strategy to lower the risks of and to overcome the year 2000 traps. In particular, it reports on experiences gained from applying preventive actions to MIS applications and keeping under control their degree of Y2K compliance. On the basis of a case study, this paper depicts some outcomes and summarizes the lessons learned.

3. BACKGROUND AND ORGANIZATION

3.1. Company background

Over the last few years, Netsiel S.p.A., a Telecom Italia-Finsiel company, headquartered in Bari (Italy) since 1988 and employing about 350 people, has been involved in large Y2K projects. Its Data Centre

Environment (DCE) provides a production environment available 24 hours a day, 7 days a week. It is powered by about 5000 MIPS of computing capacity and more than 20 Terabytes of on-line mass storage, with almost half of them committed to disaster recovery or business continuity services. Moreover, its high speed backbones, interconnecting via CDN or frame relay among more than 30 nodes all over Italy, greatly facilitate mass data exchanges among the company's sites and the sites of its customers.

As for the Y2K projects, Netsiel was particularly active in the following business sectors:

- finance;
- government;
- transportation;
- telecommunications.

This represented a unique opportunity for the company to become highly conscious of the Y2K problems of MIS applications and to acquire further experience with and a deeper insight into the organization, its business processes, and the application features in different sectors. At the same time, all the Netsiel staff involved in such projects have acquired an unparalleled experience in complex-problems management, detection and fixing of very tricky defects, contingency management, complex technological infrastructures, specific tools development and use, and application integration.

3.2. Interconnected mass change factories

From an organizational point of view, in order to satisfy any Y2K operational need, we set up several Mass Change Factories (MCF) distributed over the country and interconnected them through a centralized Technology Team (see Figure 2). This team is responsible for satisfying the new requirements for the technological infrastructure coming up from the business sites and for supporting the MCF production process.

In each MCF, the Factory Teams (FT) are responsible for the factory production, and each of them can share its own resources with other teams so to constitute a highly productive organizational model, where every engineer is acknowledged by his own competencies and responsibilities.

The members of the Business Team (BT) interface with the production teams at one end and with the customer at the other end, in order to exchange requirements and outcomes in a convenient way and to control the overall production process together with the related deadlines. Each BT member drives one or more FTs; furthermore, due to the FTs overlapping, each FT member can interact with more than one BT member at a time.

Such a network of responsibilities and relationships proved to be very effective and productive especially in complex and time-constrained tasks, as the Y2K services are. This overall organizational model was extensively adopted in the telecommunication sector, and it has greatly contributed to the successful verification or remediation of more than 100 million lines of code (LOC), running on the various target systems, ranging from the legacy MVS-COBOL systems to the Web-based ones.

3.3. IV&V approach

Among all the Y2K services, the Independent Verification and Validation (IV&V) was considered particularly relevant for 1999, so as to have it available for 2000 as one of the main preventive actions

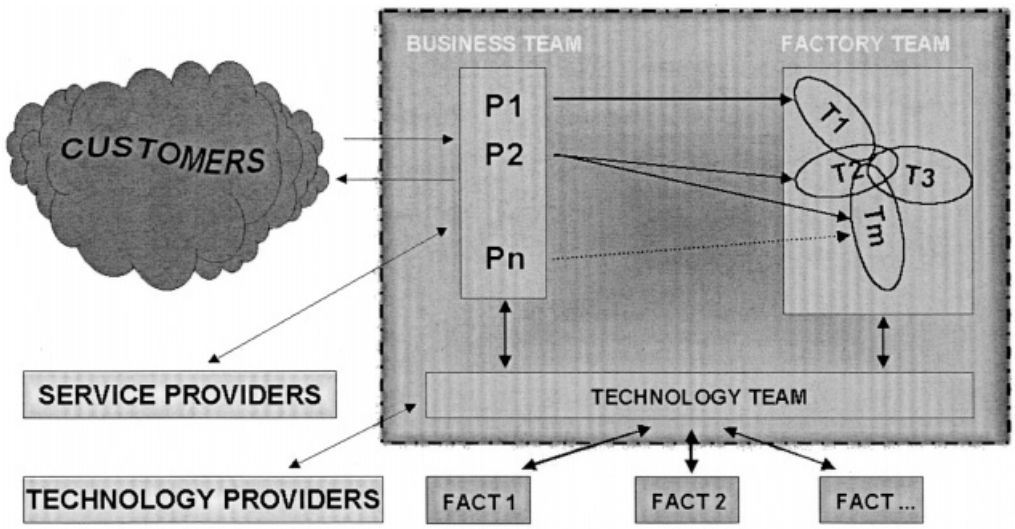


Figure 2. The organizational view of Netsiel's Mass Change Factory.

to apply to mission-critical applications, for many customers [5]. IV&V [6,7], when applied in Y2K remediation, typically consists of an independent review of the work performed on all application systems (remediated or not) in order to verify their year 2000 compliance. In particular, the remediated code is examined, before or after testing, for completeness and accuracy of the received treatments, and each system is checked for its adherence to the Y2K compliance standards, as outlined by the British Standard Institution [8].

Our approach consists of a mixture of methodology, techniques and tools focused on the application source modules. Our approach attempts to detect any remaining Y2K defects in the code. The kinds of defects that can be automatically detected while analyzing the source code include, but are not limited to, the following:

- **literal values** when using explicit century, year or date values either in initialization statements (such as `VALUE '19'`), or in procedural statements (such as `MOVE '991231' TO DT`), or as thresholds in windowing logic, are recognized and flagged in a special way so as to facilitate an analysis of their compliance status;
- **comparison statements** when involving one or more date variables or literals (such as `IF YY > 97 OR DT1 > DT2`) are specifically checked to catch potential non-compliances of variables or literals with different sizes, as for example when data from unexpanded table columns are fetched into expanded host variables;
- **computation statements** when involving one or more date variables or literals (such as `OLDY = YY-D`) are checked specifically, and leap year computations are recognized and flagged in a special way;

- **database accesses by key** when date values are part of the key are checked specifically, as in SQL, VSAM and DLI database access statements;
- **sorting statements** when the key fields include date fields are checked specifically, as in COBOL, SQL and JCL sorting statements.

All the anomalies in each of the above categories are pinpointed and well documented, and then summed up for each application so as to obtain the *application severity level* with respect to its Y2K compliance. Afterwards, this helps the client's Information Technology Units (ITU) to select, schedule and execute the most convenient fixing strategy for each application. Finally, our IV&V approach proved to be a good complement to the testing and support of the THE estimation procedure (see Section 5), so it really acts as a requirement, rather than as an option among the preventive actions of any contingency plan.

4. IV&V CASE STUDY

4.1. Context

When first approaching an IV&V process on a new application, a preliminary risk analysis task is required, so as to assess mission-criticality and prioritize subsequent interventions. As usual, and as in the case reported here, we cooperated with our customer's Information Technology Units (ITU) in order to draw a *risk map* of all their automated processes and underlying software systems. This goal was achieved through a rigorous analysis of business and technical critical issues. It took into account the most relevant risk parameters and assigned a weight to each of them. Afterwards, all the applications were checked against each risk parameter to calculate the total risk factor for each of them (see Table I). This allowed us to determine the risk map for all the information systems and to prioritize the implementation of the IV&V for each of them.

The outcomes of this task generally showed a medium level of criticality that, when matched with the customer's available budget, suggested we should apply the IV&V process to all the applications exceeding a risk level of 50. Among these applications, with a score of 80, was the *billing and credits management* (BCM) application that is the focus of this case study. Such an application really consists of several elementary applications executing on an OS/390 system. In this case study we shall consider it as a whole, where not explicitly stated otherwise. Its size is about 600K of lines of code, mainly written in COBOL and making intensive use of DB2 tables. In particular, the application consists of the following software objects:

- 360 programs;
- 224 copybooks;
- 107 DB2 tables;
- 189 JCL, PROCS and CONTROL CARDS;
- 137 screen maps.

The BCM application had previously been subjected to a Y2K remediation process that had applied the technique of expanding database date fields only where necessary in the code, as when required by sorting or accessing by key. When submitted to the IV&V process, the BCM application had not yet been listed as a candidate for the system future test.

Table I. Sample parameters and relative weights for PA prioritizing.

	Business critical	Legacy environment	Distributed environment	External date streams	Not remediated	Y2K test defects	Risk level
Priority	35	5	5	20	15	20	100
App 1	✓	✓		✓	✓	✓	95
App 2 BCM	✓	✓		✓		✓	80
App 3		✓	✓	✓	✓	✓	65
App 4	✓	✓		✓	✓		75
App 5	✓	✓		✓			55
...							
App N		✓		✓		✓	45

The factory team that was established to apply the IV&V process to this BCM application consisted of two people. Their tasks were processed in parallel whenever possible to allow us to reach to the end of the process in a couple of weeks.

The hardware platform used for this analysis included a server with double Pentium-II™ 400 MHz processors connected to several workstations, each a Pentium-II™ class computer, through a 100 Mbps LAN. Every computer ran the Windows NT™ or Windows 95™ operating system.

4.2. Process phases

The IV&V process, as defined at Netsiel's MCF, consisted of four main phases, each of them including several steps (see Figure 3). The four main phases were:

(i) *factory setup.* A factory team was established with the responsibility of conducting the whole process. The team collected the software to be analyzed, loaded each software module into the factory's computers, and categorized the software on the basis of its content. Then the modules were assigned convenient extensions, and could be pre-processed to handle programming dialects if necessary;

(ii) *inventory validation.* Source modules are parsed and a software repository containing all the relevant implicit information and relationships found in the application was built. Some reports were generated listing: (a) syntax errors; (b) missing modules; (c) duplicated modules; and (d) unused modules or inactive code (*dead code*). Complete inventory reports could have been generated too. Some support data were calculated and could be used in sizing the project;

(iii) *seeding and analysis.* Seed dates were automatically generated from I/O fields. After a validation step, they are passed through the propagation process that relies upon code and data flow analysis algorithms. All the database definitions and access schemas were analyzed in order to detect latent anomalies in implicit or explicit sorts and key field definitions;

(iv) *NC detection.* Once classes and types of anomalies had been defined together with the relative severity level we were interested in, all the detected anomalies were categorized accordingly.

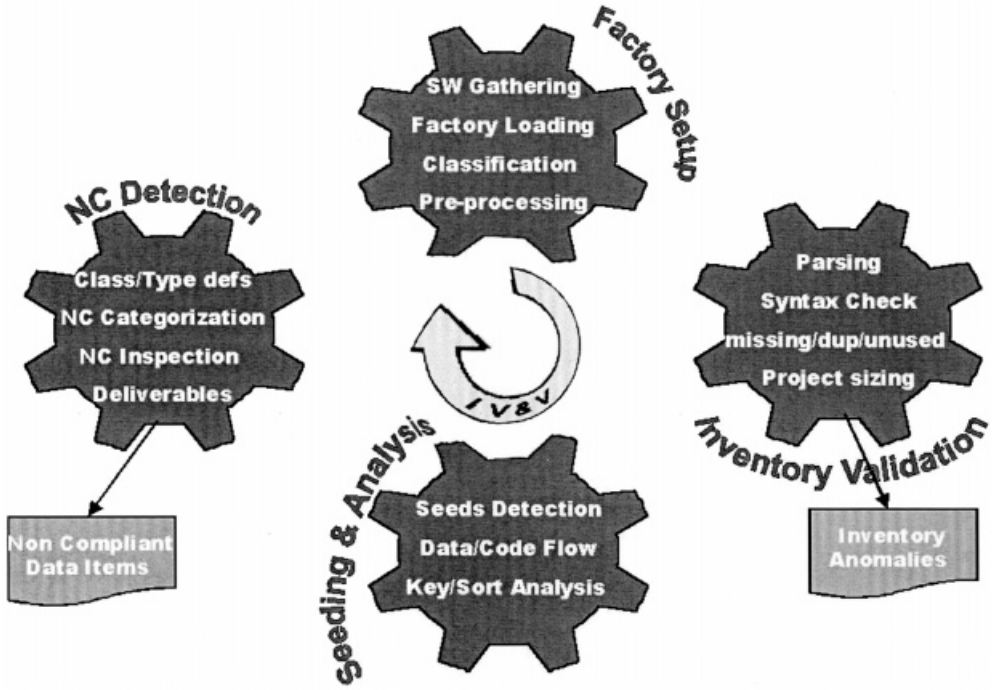


Figure 3. The IV&V process: phases and steps.

Some reports listing all the categorized anomalies were generated. Visual inspection sessions could be easily and quickly executed in order to decide whether or not to ignore some anomalies, or to assign a weight to the anomaly fixes.

All the above listed phases include adequate Quality Assurance (QA) procedures that help ensure the overall soundness and reliability of the results. Moreover, once a phase has been completed and the successor one started, the team seldom needed to go back to a previous stage in the process. This allowed management to set up meaningful milestones to aid in process management and control.

4.2.1. Phase 1—factory setup

4.2.1.1. General approach. In this phase we established a Factory Team consisting of two trained and skilled engineers. The team had to collaborate with the customer in order to locate all the relevant and active source modules in their actually used versions on the host system, copy them into the MCF's computers, and then conduct the whole IV&V process. The extraction of the source modules in a suitable format and their loading into the factory's computers proved to be an automated and straightforward operation for the MCF staff. This was mainly possible thanks to a fruitful cooperation

with the customer's ITU and the soundness of the technological infrastructure (see Section 3.1). However, it is important to note here that such an apparently simple operation can sometimes become bogged down with a substantial loss of time.

During the categorization of the software modules, some of them were not automatically recognized due to their erroneous library or directory placements—e.g., some copybook modules in the program library. This situation required the relocation of some modules. When called to the attention of the ITU and fixed by the ITU, it allowed the ITU to reorganize the customer's original software libraries and to reestablish the correct configuration control over those BCM software assets. Such improvements have been identified as an important benefit from well-done Y2K remediation [9].

In the BCM, no specific programming dialects or extensions to the standard programming languages were present, so no special handling was required.

4.2.1.2. Deliverables. The main deliverable from this phase was the establishment of the BCM application's updated and categorized source modules into the MCF's mass-computing environment. Each module of a given type was assigned its own filename extension and was placed into a specific directory, as for example, COBOL programs into the '. . . \ COB' directory with the '.COB' extension, copybooks into the '· · · \ CPY' directory with the '.CPY' extension, etc.

Moreover, the list of module relocations in the customer's production environment was generated and provided to the customer so as to update the original software libraries on a consistent basis.

4.2.2. Phase 2—inventory validation

4.2.2.1. General approach. The application inventory task in the IV&V process is sometimes considered a waste of time or an optional step. Nevertheless, in many cases it allows the team to discover new modules that otherwise would have never been checked by any Y2K verification process. Here are some of the causes:

- when the inventory was performed at the beginning of the remediation process, some of the modules belonging to the application could have not been detected as relevant;
- during the remediation process, some modules may not have been considered or erroneously discarded or temporarily put aside in order to be subsequently processed, and then neglected;
- at the end of the remediation process, some modules may not have been sent back to the test environment as part of the final software configuration.

Moreover, the new modules added to the application by the concurrent or ongoing maintenance could prove not to be fully Y2K compliant, thus necessitating a new verification process. This well-known problem, sometimes referred to as the *Compliance Level Maintenance* problem, usually arises where a weak software configuration control exists. In the BCM, we parsed all the source modules so to populate the application repository with the relevant information about data definitions, their uses in the program execution flow, and all the cross-references among the source modules.

Once the parsing process had been completed, we decided to split the project into two streams or frames, in order to process them in parallel during the subsequent phases. In order to make such a split easy to do and to make an unequivocal boundary between the two streams, we decided to separate from the rest of the modules all of the software modules related to the on-line transactions.

Table II. Summary of inventory issues.

Type of issue	Occurrences	Modules impacted	Notes
Syntax errors	35	6	
Missing objects	4	114	
Unreferenced objects	32	N/A	All copybooks
Duplicated objects	0	0	

This gave us some advantages. From our estimations, this split lowered the overall project time by about 15% and allowed us to execute more accurate QA *cross-checks* to gain higher quality in the team's performance.

4.2.2.2. Deliverables. The reports generated in this phase provided data about the following topics:

- syntax errors/warnings;
- missing objects;
- unreferenced objects;
- duplicated objects.

It took about two hours to perform the BCM software inventory, to scan the source code, and to collect all the required data (see Table II).

As for the syntax issues, we found six modules containing a total of 35 syntax anomalies, although the BCM application had previously been subject to a Y2K remediation process. Actually, it can also happen that even the best correction processes and tools might fail when up against particularly complex and unexpected situations. As shown in Figure 4, the syntax issues are classified by their severity level (notice, warning and error), and are then grouped by similar occurrences in different modules so as to speed up the issues analysis and the subsequent code fixing, where required.

The values for the rest of the inventory results were pretty good, considering that the missing objects were due to *generalized routines*. These routines did not require any further verification, having been comprehensively tested during the prior BCM application remediation. Moreover, a few copybooks were found in excess and they were simply ignored during the rest of the process. In this case, we decided not to perform any *dead-code analysis* on the BCM application software, because the customer did not consider it to be critical.

As a final deliverable of this phase, all the cross-references among the software objects in the BCM application were accessible from our BCM application repository. This repository may be queried at any time in order to collect references for each single object or for a group of objects, a very helpful facility in the later phases.

Figure 4. Example of list of syntax errors/warnings.

Table III. Summary of seeds list.

Type of seed	Elementary	Composite	Modules impacted
I/O fields	491	3	65
Local variables	6450	470	299
Total	6941	473	364

4.2.3. Phase 3—seeding and analysis

4.2.3.1. General approach. Seeding is often the critical step in an automated Y2K remediation process. The same is obviously true for IV&V. Since seeding is an iterative process, its convergence and quality of results are dependent upon the quality of the supporting tools. In particular, the seeding tool has to implement a robust algorithm in order to locate all the date fields, minimizing the sets of false positives and false negatives. This can be only achieved by using the following techniques:

- implementing rigorous pattern-matching (i.e., *regular expressions*);
- checking the record structures and composite fields (field re-definitions are included);
- verifying the initialization values; and
- verifying the semantics of fields used in the same statement (through fast and reliable *seeds propagation*).

Beside the implementation of the above techniques, we also applied a smart propagation algorithm, capable of helpfully integrating the *data-flow* and the *code-flow* analyses. Going further in these analyses, we were able to detect database fields definitions containing references to dates, and to analyze the database accesses in order to find dates used in key fields or as sort keys.

4.2.3.2. Deliverables. The outcomes of the seeding process consisted of a list of date-related variables along with the associated semantics. Each date variable was highlighted along with the memory displacement where the year data are stored, and the century, if any. The seeds list may consist of either I/O fields only—e.g., record fields, DB2 table columns, etc.—or every date-related variable declared locally within each program.

Figure 5 shows some of the search patterns used, in regular-expression format, together with a partial list of the initial seeds. In particular, the seeds are coded for each row with the variable name, the `PICTURE` of the COBOL field, its semantics (e.g., yy as year, dd/mm/yy as formatted date, cc as century, etc.), and the locations in the programs or copybooks where the variable is defined.

In our case, about 83% of the programs in the application were impacted, though a low percentage (as low as 7%) of I/O fields deal with date values, as shown in Table III. The column of composite variables represents those variables containing date data not only, but counters, flags and the like.

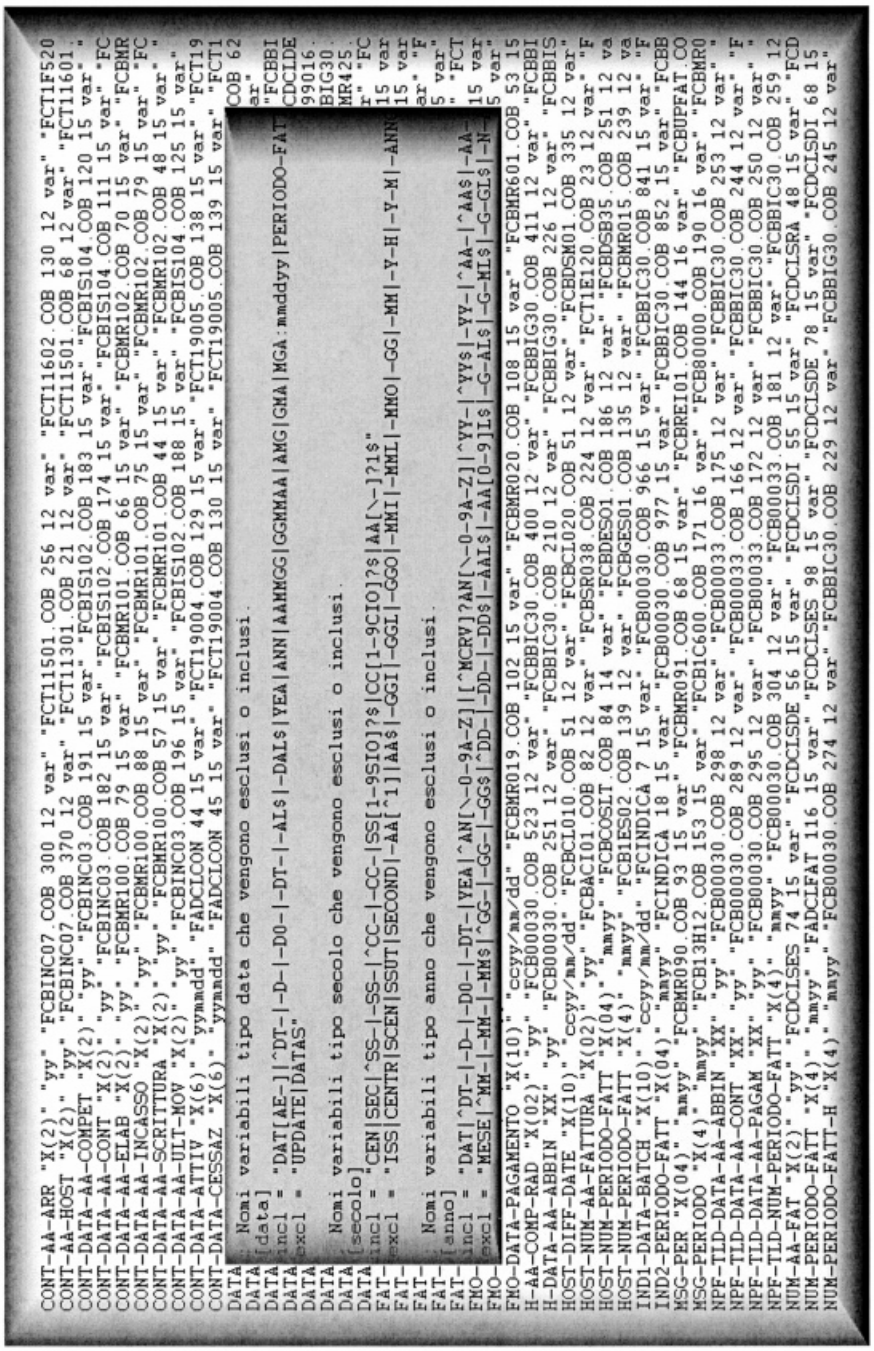


Figure 5. Search patterns and initial seeds.

4.2.4. Phase 4—NC detection

4.2.4.1. General approach. Once all dates and related uses had been pinpointed inside the source modules, the time had come to find the anomalies or *Non-Compliances* (NC). First of all, we established the *compliance criteria* by defining several classes of Non-Compliance, as listed in Section 3.3. In particular, these represent the basic compliance criteria in accordance with the year 2000 compliance definition of the British Standard Institution [8]. They can be customized in order to satisfy nearly any compliance verification need thanks to the great flexibility of the underlying analysis tools. Moreover, each class can be assigned to a *priority level* to allow the tools to automatically solve multiple issues on the same statement, by picking up the one with the highest priority.

At the end of this process, every anomalous use of date-related statements was accordingly categorized into a list of actual and potential NCs. In particular, the latter needed to be further inspected in order to minimize the *false positives*. Such a task sometimes requires contributions from the application domain experts, especially when the issues are related to *business rules* or application logic knowledge.

Conversely, in our case the factory team did not require any contribution by the BCM application domain experts. Its work was facilitated by the use of the Date Sensitive Browser (DSB), a proprietary tool which speeded up the inspection sessions through synchronized navigation capabilities in the source code and the data-flow analysis results.

4.2.4.2. Deliverables. The final deliverable of the IV&V process consists of a list of non-compliances. As shown in Figure 6, the listing of the anomalies is by module and included a unique identifier local to the module for further references, and trailing information. Among the trailing information, the following items of data are noteworthy:

- statement type—e.g., lt, gt, subtract, move, value, etc.;
- NC class with its priority in square brackets—e.g., COMPARISON-CONST, SRTKEY-YEAR, etc.;
- involved variables or constants, with the related semantics in square brackets; and
- the full line of code, following the dots '. . . '.

As the example shows, the IV&V tools proved to be very effective in detecting such non-compliant treatments as date movements and comparisons involving substring indexes, string concatenations with century constants, and complex SQL selection statements with implicit ordering on date fields. Thanks to the effectiveness and soundness of the IV&V tools, even more tricky anomalies were normally spotted in a relatively short time.

Once all the NCs had been detected for each processed application, we were able to estimate their *severity level* in terms of effort needed to make the application compliant. As reported in Figure 7, each anomaly received a severity level assignment (i.e., low, medium or high) on the basis of the class it belongs to and the corresponding effort required to fix it. Afterwards, by summing up the anomalies in each category and setting up empirical thresholds, we got the severity level for the whole application. Most of the BCM components were classified at the high severity level. The severity rating proved to be very useful for the customer's ITU, in order to properly schedule and size the fixing tasks.

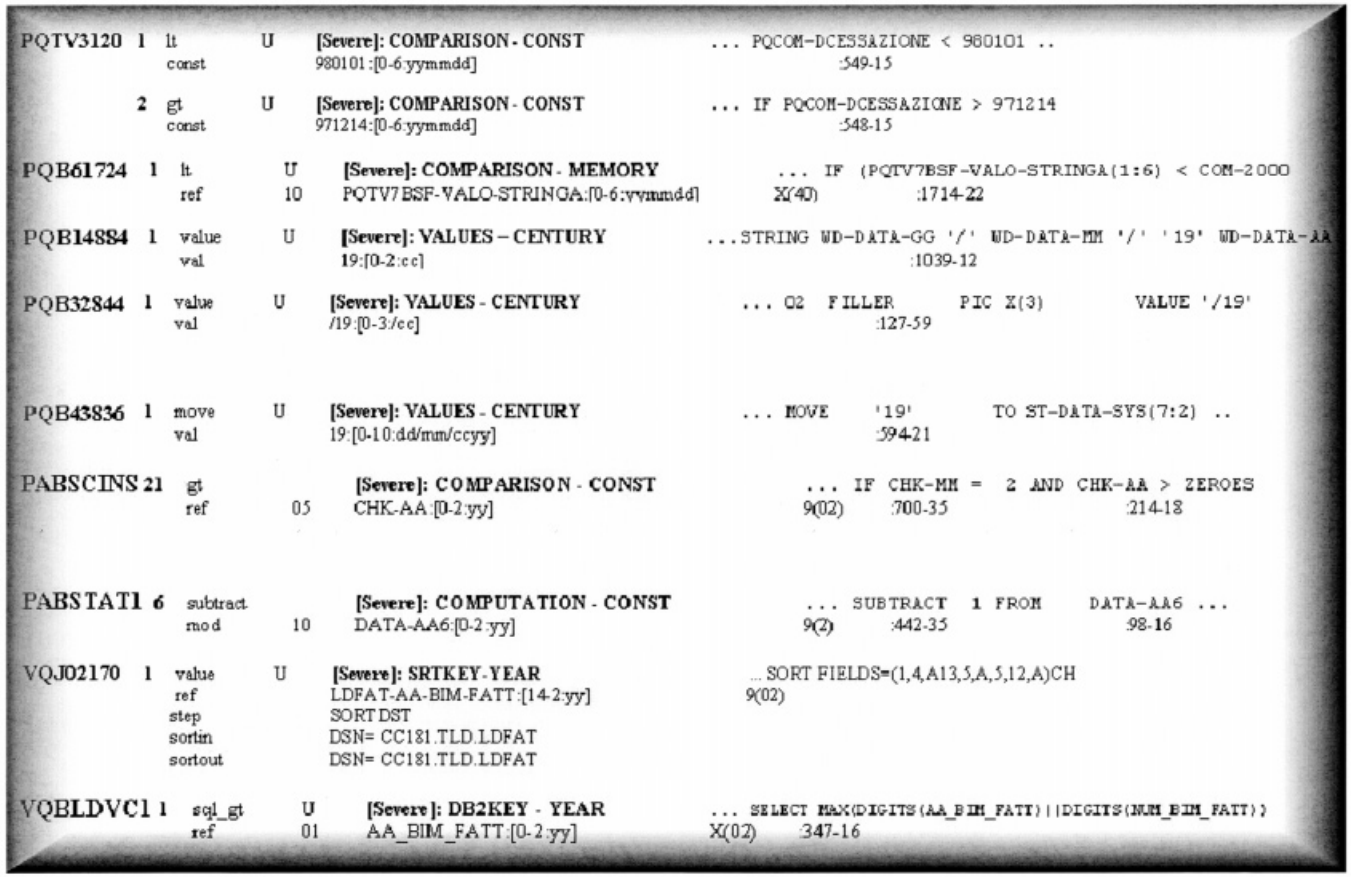
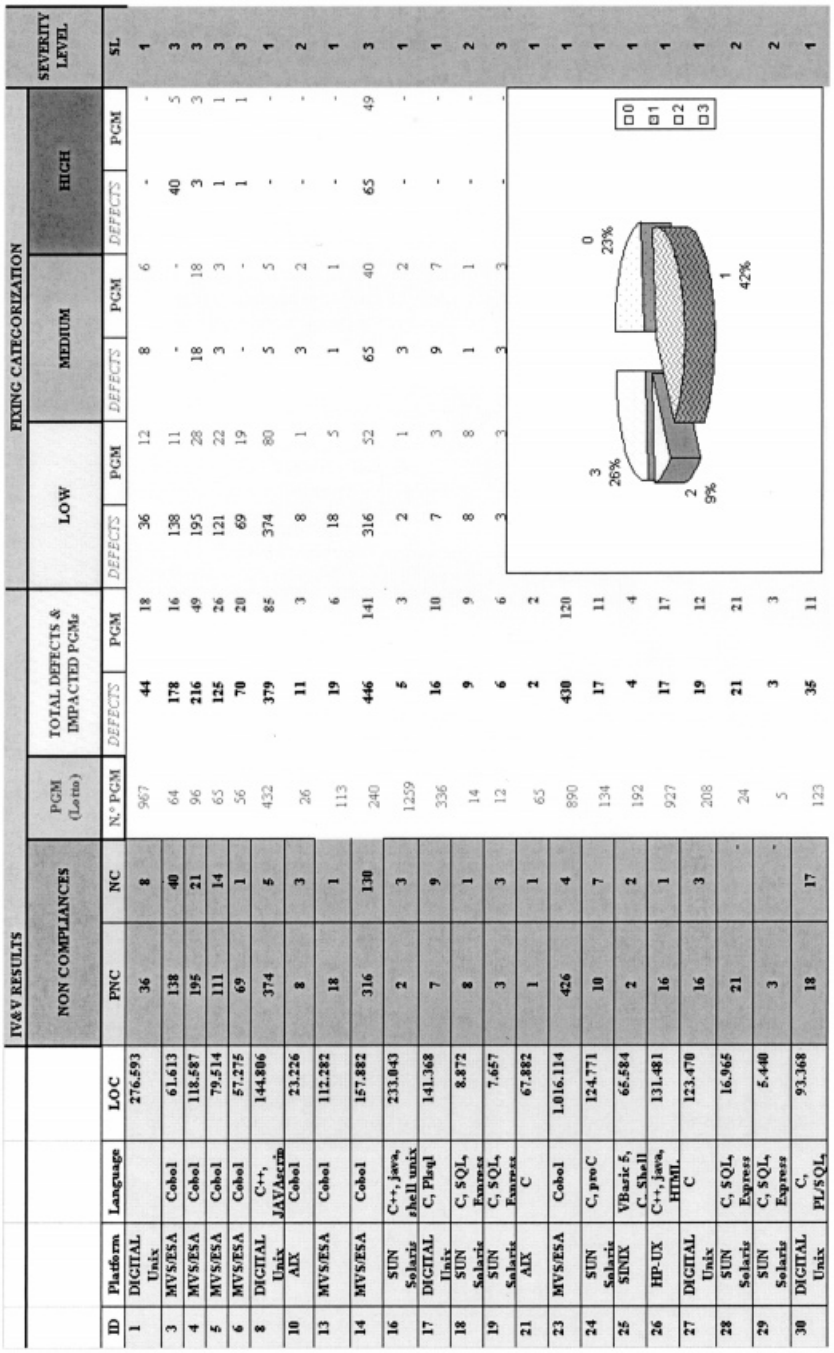


Figure 6. List of actual Non-Compliances.



			IV&V RESULTS			FIXING CATEGORIZATION								SEVERITY LEVEL
			NON COMPLIANCES		PGM (Lotto)	TOTAL DEFECTS & IMPACTED PGMs		LOW		MEDIUM		HIGH		
ID	Platform	Language	LOC	PNC	NC	N.° PGM	DEFECTS	PGM	DEFECTS	PGM	DEFECTS	PGM	DEFECTS	PGM	SL
1	DIGITAL Unix		276.593	36	8	967	44	18	36	12	8	6	-	-	1
3	MVS/ESA	Cobol	61.613	138	40	64	178	16	138	11	-	-	40	5	3
4	MVS/ESA	Cobol	118.587	195	21	96	216	49	195	28	18	18	3	3	3
5	MVS/ESA	Cobol	79.514	111	14	65	125	26	121	22	3	3	1	1	3
6	MVS/ESA	Cobol	57.275	69	1	56	70	20	69	19	-	-	1	1	3
8	DIGITAL Unix	C++, JAVAscrip	144.806	374	5	432	379	85	374	80	5	5	-	-	1
10	AIX	Cobol	23.226	8	3	26	11	3	8	1	3	2	-	-	2
13	MVS/ESA	Cobol	112.282	18	1	113	19	6	18	5	1	1	-	-	1
14	MVS/ESA	Cobol	157.882	316	130	240	446	141	316	52	65	40	65	49	3
16	SUN Solaris	C++, java, shell unix	233.043	2	3	1259	5	3	2	1	3	2	-	-	1
17	DIGITAL Unix	C, Plsql	141.368	7	9	336	16	10	7	3	9	7	-	-	1
18	SUN Solaris	C, SQL, Fastest	8.872	8	1	14	9	9	8	8	1	1	-	-	2
19	SUN Solaris	C, SQL, Fastest	7.657	3	3	12	6	6	3	3	3	3	-	-	3
21	AIX	C	67.882	1	1	65	2	2							1
23	MVS/ESA	Cobol	1.016.114	426	4	890	430	120							1
24	SUN Solaris	C, proC	124.771	10	7	134	17	11							1
25	SINIX	VBasic 5, C, Shell	65.584	2	2	192	4	4							1
26	HP-UX	C++, Java, HTML	131.481	16	1	927	17	17							1
27	DIGITAL Unix	C	123.470	16	3	208	19	12							1
28	SUN Solaris	C, SQL, Express	16.965	21	-	24	21	21							2
29	SUN Solaris	C, SQL, Express	5.440	3	-	5	3	3							2
30	DIGITAL Unix	C, PL/SQL	93.368	18	17	123	35	11							1

Figure 7. Defects categorization and fixing severity estimation.

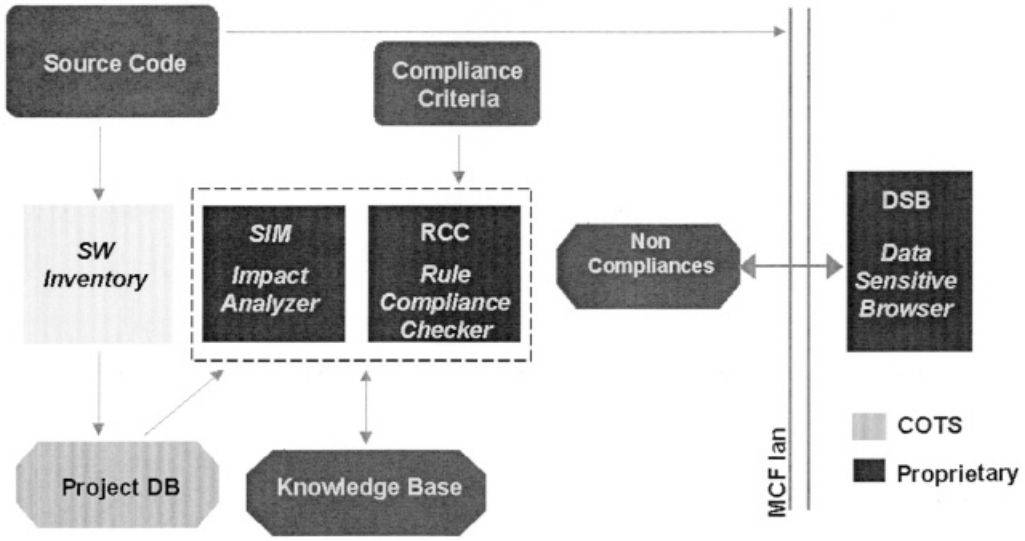


Figure 8. Tools architecture.

4.3. Benefits achieved

The major benefits achieved by our approach were:

(i) reduced overall project elapsed time, thanks to effective process management and the availability of supporting tools;
(ii) enhanced quality of the deliverables, especially in the reduction of the number of *false positives* and *false negatives* present in the potential NCs list (the quality of the final and intermediate deliverables was aided by specific checks applied at the end of each phase and by *cross-checks* between the transactional application stream and the batch application stream);
(iii) speeding up of the fixing of anomalies, with right-sized fixing strategies (the NC categorization into severity levels, and the classification of the whole BCM application itself at the high severity level, allowed the customer to apply appropriate resources in order to fix the anomalies in the shortest time possible); and
(iv) speeding up of the test phase (the outcomes of the IV&V process allowed the customer greatly to reduce the resource-intensive future-test effort and to concentrate the future-test on most critical anomalies that we detected).

4.4. Supporting tools

In order to provide an automated support to the IV&V process and to make the job of the factory team cheaper, easier and faster in each phase of the process, we built up a technological infrastructure

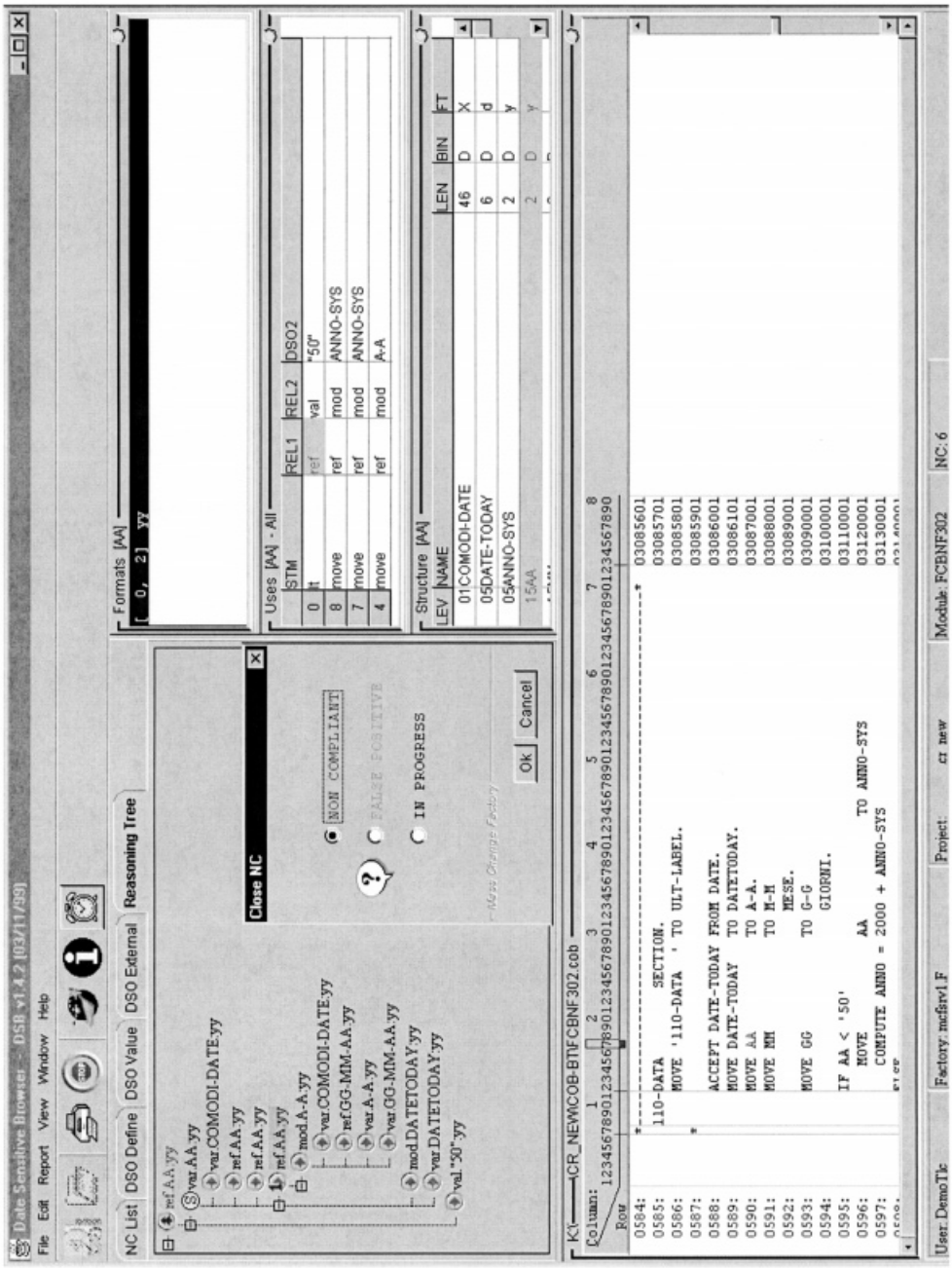


Figure 9. An example screen from the Date Sensitive Browser (DSB).

including both leading-edge COTS (commercial off-the-shelf) tools and proprietary technologies (see Figure 8). These greatly contributed in the optimization of the process steps and the quality of the final results.

In particular, among the COTS tools, the following were noteworthy:

- Revolve™ (from Merant), a source code scanner and application repository; and
- Crystal Reports™ (from Seagate Software), a high-quality report generator.

In the proprietary tools set [10] were:

- PACKDAT, a smart and very fast tool for the automatic detection of date fields and related formats on the basis of the specified criteria;
- SIM/RCC (Rules Compliance Checker), a data-flow analysis tool with rule-based features for automatic anomalies detection;
- DSB (Date Sensitive Browser), a powerful distributed Java application, very helpful for anomalies inspection, with synchronized views on the source code and the data-flow analysis results (see Figure 9).

Further details about the tools used for each process step can be found in an earlier paper [11].

5. FURTHER PREVENTIVE ACTIONS

The IV&V process only represents one, though a fundamental one, of the possible preventive actions that should be taken in the year 2000 in order to lower the risk of residual Y2K failures and meet the due diligence requirements. But it is not the only one. In fact, there are at least a couple of other techniques that might be considered: the System Future Test (SFT) and the Time Horizon of Events (THE) [12].

The first consists of defining specific Y2K test cases and executing them in a Y2K test environment, distinct and isolated from the production environment. Such a test environment needs to be consistently *aged* to cover a span of future calendar time. To do this requires that the dates stored in the database and all the sources providing the software applications with date values must use the future dates. Moreover, the internal integrity of all the records stored in the database still needs to be preserved. We tried using such a technique extensively in the last months of 1999, and often found the process to be more complex and expansive and slower than expected, and it seldom covered all the real cases. During the year 2000, we anticipate its applicability will be even more complex.

The THE relies upon the concept of periodically monitoring the date data stored into the application database in order to estimate when a given date, such as 01/01/00, 30/03/2000, etc., comes into the database, in the production environment. In fact, each date field has its own time horizon. What we really need to know is what these horizons are in order to calculate the failure time likelihood for each of them. Such a technique is particularly useful in order to anticipate how the failure likelihood changes for all the date fields in a database as time advances.

When combined with the IV&V outcomes, this technique becomes very powerful. In fact, in such a context, it allows us to cross-reference the database fields suspected for failures in the short term with the application programs accessing them. Moreover, the I/O seeds and the non-compliances detected

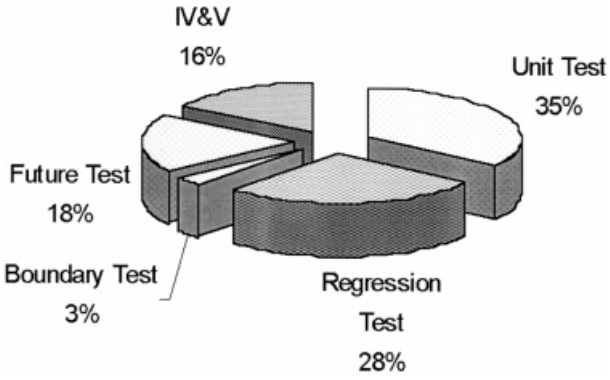


Figure 10. Distribution of anomalies detection.

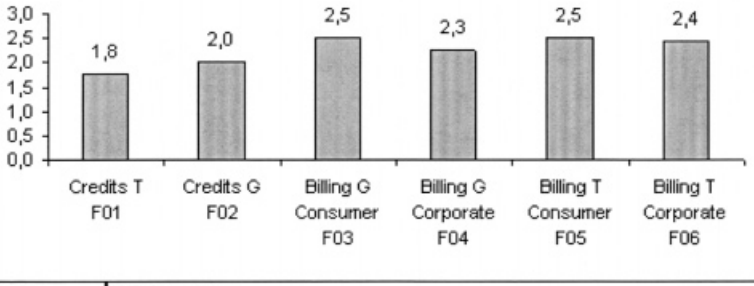
during the IV&V process can be further qualified with estimates of their actual *short-term risk level*. Tedone [13] offers a detailed report on the THE technique.

In Figure 10, a graph depicts some statistics that we computed on the basis of the production data collected at Netsiel's Mass Change Factory. In particular, it shows the percentage of the anomalies detected at the end of each test phase and at the end of the IV&V process. The assumption is that each test is performed in the sequence shown and that the IV&V process is performed after the system future test completion. Moreover, the application is assumed to have been previously remediated by some automated Y2K anomaly fixing tool. What needs to be highlighted here is that after an application has been remediated and tested, residual (i.e., remaining unremoved) Y2K anomalies averaging about 16% of the original Y2K anomalies can still be found by using an IV&V process. Our observation of residual anomalies is higher than the less than 10% forecast from 1998 noted earlier [2]. Also, we observe the percentage results to be much higher in the situation where the application has been manually remediated, since we observe that in such a situation the system future test percentage drops.

6. LESSONS LEARNED AND FUTURE STEPS

Drawing upon the experience reported in this 1999 case study and the much wider experience of the last few years in Y2K projects, this paper has highlighted the strategic role of preventive actions in any Y2K contingency plan for use during the year 2000. In particular, this paper has laid out the central role of the IV&V process among the preventive actions, showing how it can effectively complement the testing tasks.

Now, since the 1 January 2000 has come and gone, we cannot expect that the *millennium bugs* have gone with it. In fact, many of them are still present in the MIS systems in a latent state, and may reveal their presence anywhere in an unpredictable way. What is worse, they can cause unrecoverable damage to the data assets of companies worldwide if they are not discovered in time.



Code	BP Instance	BP Intervention Level	# Involved Systems	Sum of Intervention Levels for each system	Minimum Intervention Level for a system
F01	Credits T	1,8	4	7	1
F02	Credits G	2,0	5	10	1
F03	Billing G Consumer	2,5	4	10	2
F04	Billing G Corporate	2,3	8	18	2
F05	Billing T Consumer	2,5	4	10	2
F06	Billing T Corporate	2,4	7	17	2
	....				

Figure 11. Intervention level distribution for Business Processes (BPs).

In order to avoid this situation or to reduce the risks of it, we believe that in addition to the *consequential actions* that each company usually embraces in its own contingency plan, much more attention should been given to the *preventive action*s that can be taken. This is the main lesson learned. In particular, in relation to the mission-critical Business Processes (BPs), the preventive actions should be concentrated on those BPs with a *lower intervention level*. The intervention level is given by the average of the intervention levels of the component software applications (see Figure 11, as an example). In turn, the intervention level of an application depends upon the cumulative number of Y2K remediation or preventive actions that have been performed upon its component systems.

In particular, combining the IV&V and the THE processes should be considered as a proactive and economically attractive solution to reduce the risks of Y2K failures and the consequential recovery costs.

Finally, we believe that the experience acquired in the management of complex projects, as most of the Y2K projects have been, will be a foundation for approaching similar challenging problems, like the euro transition and the systems migration from legacy to e-commerce. Moreover, the *application management* area [14] will widely benefit from the Y2K experience, especially in terms of software engineers' expertise, configuration management, application knowledge and *preventive maintenance*, until now often dismissed as a waste of time and money.

ACKNOWLEDGEMENTS

This paper is based on data coming from several IV&V and Y2K projects implemented at the Netsiel's mass change factory. Special thanks are due to the Finsiel-TLC and Netsiel-MCF engineers who collected most of the supporting data (too long a list to mention all the names here) or who provided the author with some precious insights about the paper's general form. The author also thanks the referees for their observations and suggestions.

REFERENCES

1. Gartner. *Year 2000 World Status, 2Q99: The Final Countdown', Strategic Analysis Report.* Gartner Group: Stamford CT, 1999. Also available at URL: http://www.gartner.com/public/static/y2k/00081966.html [28 October 1999].
2. Gartner. *Year 2000: Independent Verification and Validation Market, Research Note SPA–06–8620*. Gartner Group: Stamford CT, 1998.
3. Aberdeen. *Software Automation Tools Augment Year 2000 Contingency Planning White Paper*. Aberdeen Group: Boston MA, 1999. Also available at URL: http://www.aberdeen.com/ab%5Fabstracts/1999/04/04991409.htm [28 October 1999].
4. GAO. *Year 2000 Computing Crisis: Business Continuity and Contingency Planning, GAO/AIMD-10.1.19.* General Accounting Office: Washington DC, 1998. Also available at URL: http://www.gao.gov/special.pubs/bcpguide.pdf [28 October 1999].
5. INFOA. *The Final Year 2000 Test: Independent Verification and Validation White Paper*. Information Analysis Inc.: Fairfax VA, 1998. Also available at URL: http://www.infoa.com/iv&v.html [28 October 1999].
6. Arthur JD, Gröner MK, Hayhurst KJ, Holloway CM. Evaluating the effectiveness of independent verification and validation. *IEEE Computer* 1999; **32**(10):79–83.
7. LaMarr Y, Fravel WE. Software independent verification and validation: a process perspective. In *Proceedings of the Conference on Ada*. ACM Press: New York NY, 1991; 408–417.
8. BSI. *A Definition of Year 2000 Conformity Requirements, BSI-DISC PD2000.1*. British Standard Institution: London, UK, 1998. Also available at URL: http://www.bsi.org.uk/disc/2000.html [28 October 1999].
9. Marcoccia LJ. Building infrastructure for fixing the year 2000 bug: a case study. *Journal of Software Maintenance* 1998; **10**(5):333–352.
10. Ludovico C. Strumenti e servizi per arrivare al cambio del secolo con i numeri giusti. *Toolnews* 1999; **7**(2):22–25. Also available at URL: http://www.itware.com/tool9902/22_23.htm [28 October 1999].
11. Interesse M, Dabbicco R. Beyond year 2000 remediation: the compliance verification—a case study. In *Proceedings of the International Conference on Software Maintenance*. IEEE Computer Society Press: Los Alamitos CA, 1999; 155–160.
12. Gartner *Year 2000 Boundary Management, Information Technology Symposium 1999, ESC11Y2Kbound1199Akyte*. Gartner Group: Stamford CT, 1999.
13. Tedone P. *The Radar Effect White Paper 1998/AZ156v2/BMT/tr*. HAL S.p.A.: Milan, Italy, 1998. Also available at URL: http://www.radar2000.com/radar_us/mvs/htm/document.htm [28 October 1999].
14. Campobasso A, Interesse M. Application management: an industrial repository-based approach to software maintenance. In *Proceedings of the 4th European Conference on Software Maintenance and Reengineering*. IEEE Computer Society Press: Los Alamitos CA, forthcoming.

AUTHOR'S BIOGRAPHY

Michelangelo Interesse is responsible for the continuous improvement of the application management platforms at Netsiel's Software Maintenance Competence Center. He is primarily concerned with analyzing software maintenance requirements, defining architectural solutions and evaluating, selecting and integrating leading-edge technologies in the application management field. He has been working on the year 2000 situation in software since 1995, when he started to conduct the development project for a tool set for analyzing legacy applications, which now represents the core technology of the IV&V service. He has been author and co-author of a number of papers, ranging from robotics to software maintenance, since 1988. His email address is: m.interesse@netsiel.it

